

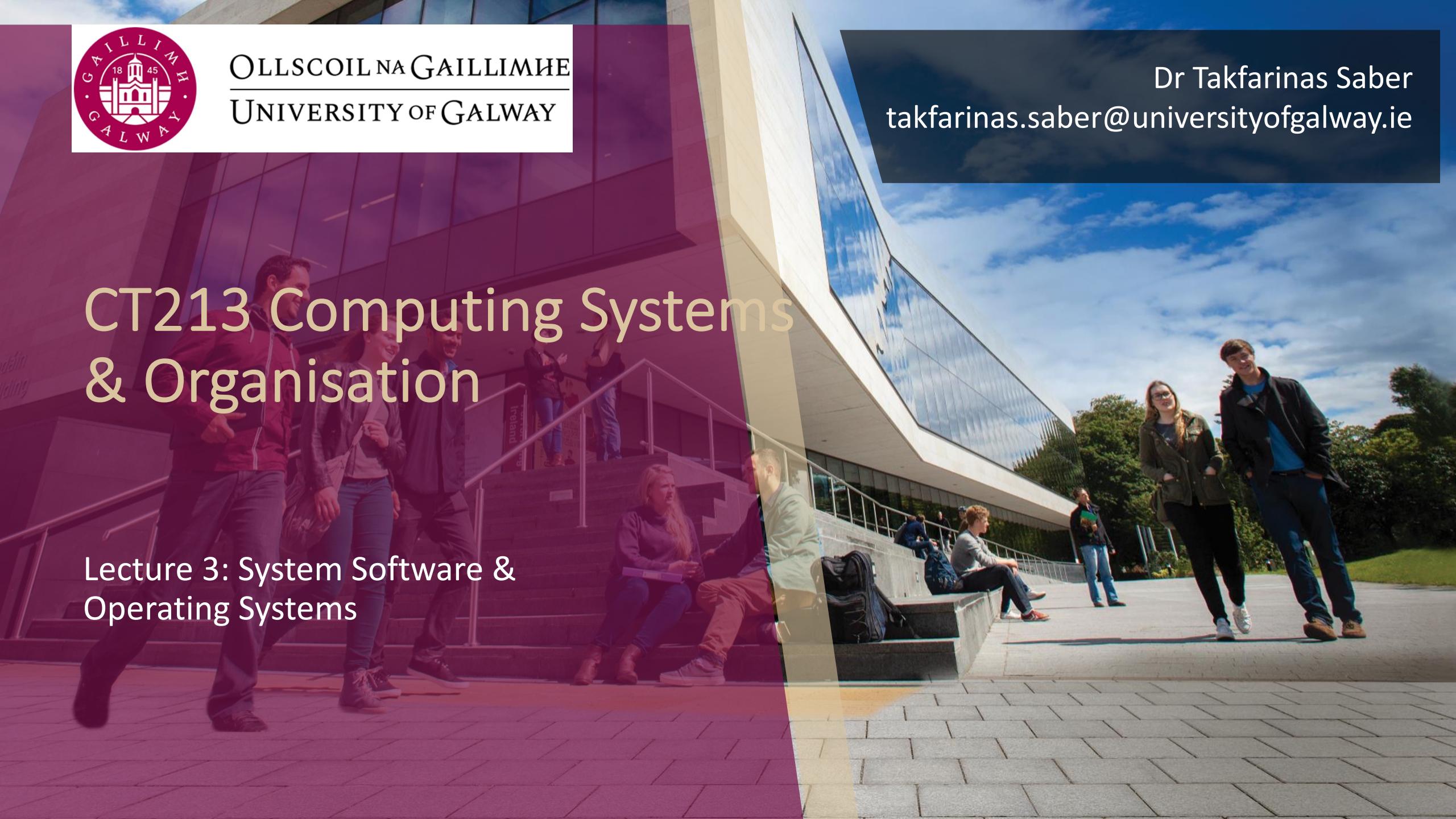


OLLSCOIL NA GAILLIMHE
UNIVERSITY OF GALWAY

CT213 Computing Systems & Organisation

Lecture 3: System Software &
Operating Systems

Dr Takfarinas Saber
takfarinas.saber@universityofgalway.ie



Contents

- System Software & OS
- OS Organisation
- OS Design and Implementation
- Implementation considerations
 - Processor modes
 - Kernel
 - Requesting services from OS

System Software & OS



Application Software

- A computer program designed to **perform a group of coordinated functions** for the benefit of the user
- **Application software** is designed to solve a specific problem
- Examples of an application include word processor, a spreadsheet, an accounting application, a web browser, photo editor etc.



System Software

- Programs dedicated to **managing the computer**
- System software is a software that **provides a platform** to other software.
- **System software** provides a general programming environment
- There are two main types of system software
 1. Operating System
 2. Utility Software



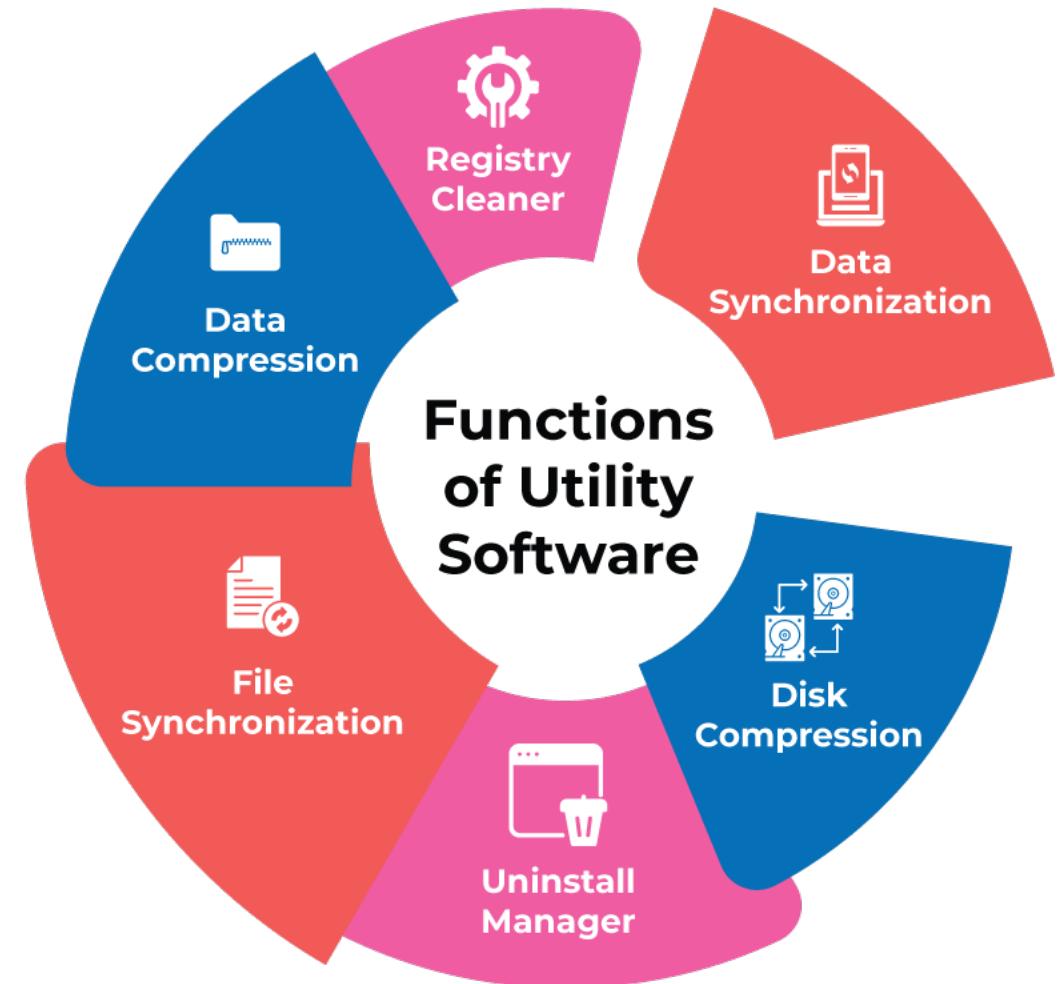
Operating System (OS)

- Provides functions used by the **application software**
- Provides the mechanisms for application software to **share** the hardware in an orderly fashion to:
 - **increase the overall performance** by allowing different application software to use different parts of the computer at the same time
 - **decrease the time to execute** a collection of programs and **increase overall system performance**
- Interacts directly with the hardware to provide an interface to other system software and with application software whenever it wants to use system's resources
 - It is application-domain independent
 - Provides resource abstraction
 - Provides resources sharing (through strict resource management policies)



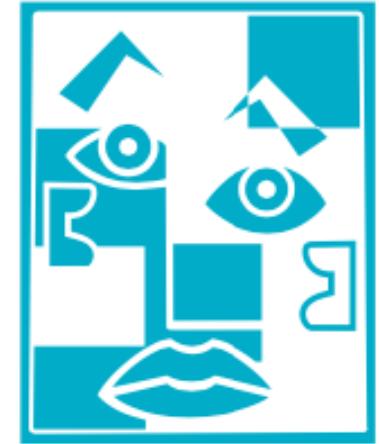
Utility Software

- Utility software is system software designed to help **analyse, configure, optimize or maintain** a computer.
- Examples of this include:
 - data compression
 - disk cleaners
 - disk defragmentation
 - registry cleaners
 - system monitors



Resource Abstraction

- It is done by providing an **abstract model of the operation** of the hardware components
 - Different hardware components that a program may access are referred to as **resources**.
 - Any particular resource, such as a hard-disk has a generic interface that defines how the programmer can make the resource perform a desired operation.
- Abstraction generalises the hardware behaviour but restricting the flexibility
 - With abstraction, certain operations become **easy to perform**, other may become **impossible** (such as specific hardware control)
- An abstraction can be made to be much simpler than the actual resource interface
 - Similar resources can be abstracted to a common abstract resource interface
 - E.g., system software may abstract hard-disks and CD-ROMs into a single abstract disk interface

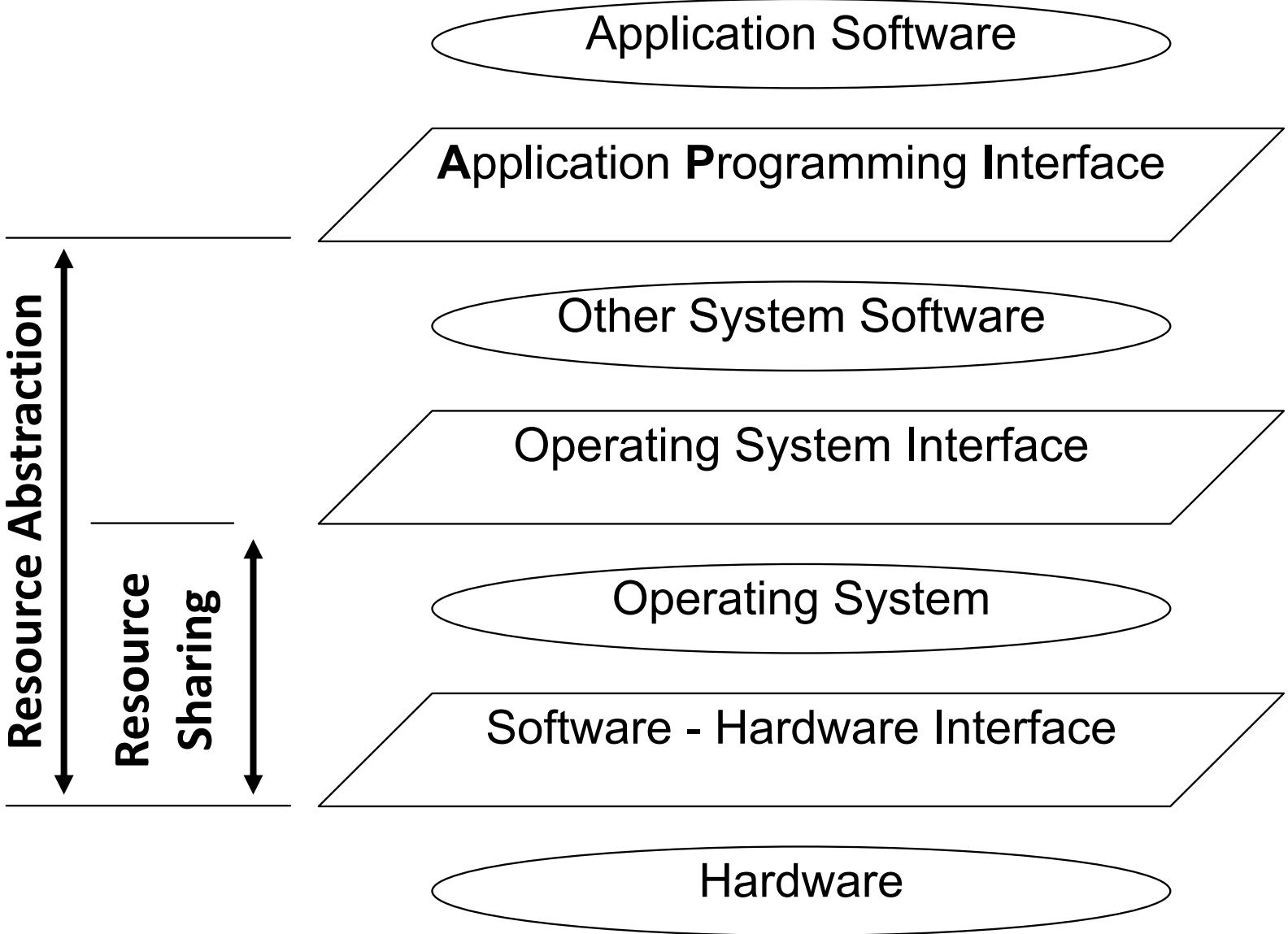


Resource Sharing

- Abstract and physical resources may be shared among a set of concurrently executing programs:
- **Space Multiplex Sharing**
Resource can be divided in two or more *distinct units* of the resource that can be used independently.
E.g.: Memory, HDD
- **Time Multiplex Sharing**
A process is allocated exclusive control of the entire resource for a short period of time (*not spatially divisible*)
E.g.: Processor Resource



System Software and the OS



OS Organisation

Process and resource manager

- It uses the abstractions provided by the other managers
- Handles resource allocation

Memory manager

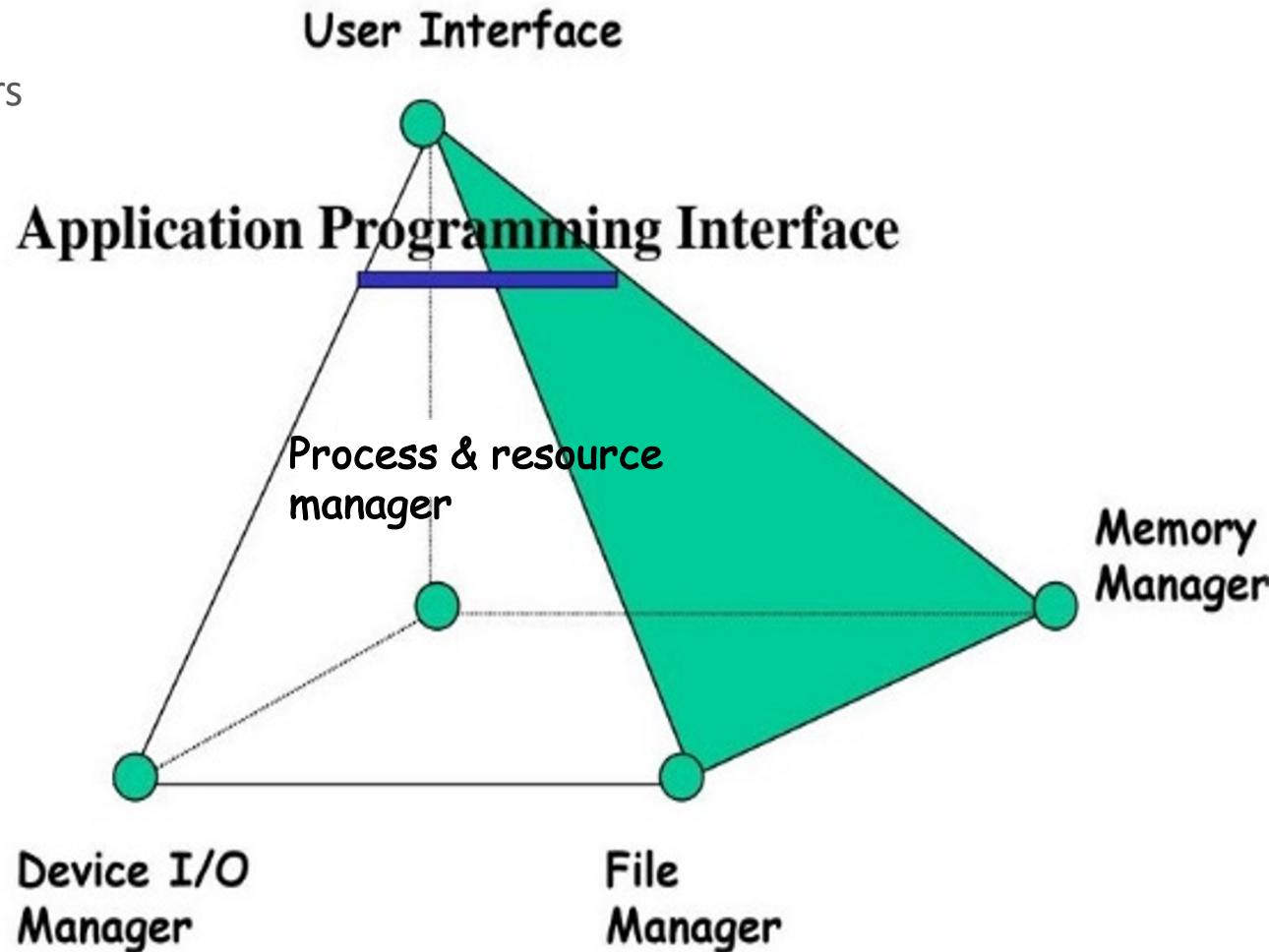
- It is classically a separate part of the operating system
- Beside other functions, it is in charge with the implementation of the virtual memory

File manager

- abstracts device I/O operations into a relatively simple operation

Device manager

- handles the details of reading and writing the physical devices
- implemented within device driver



OS Design – Functional Requirements

Processes:

- Creation, termination, control, exception handling
- Protection
- Synchronisation and communication
- Resources allocation/de-allocation

File System Management:

- Space allocation/de-allocation
- Protection, sharing, security
- Physical resource abstraction

Memory management:

- Allocation/de-allocation
- Protection and sharing

I/O devices:

- Allocation/de-allocation
- Protection and sharing
- Physical resource abstraction



Operating Systems Evolution



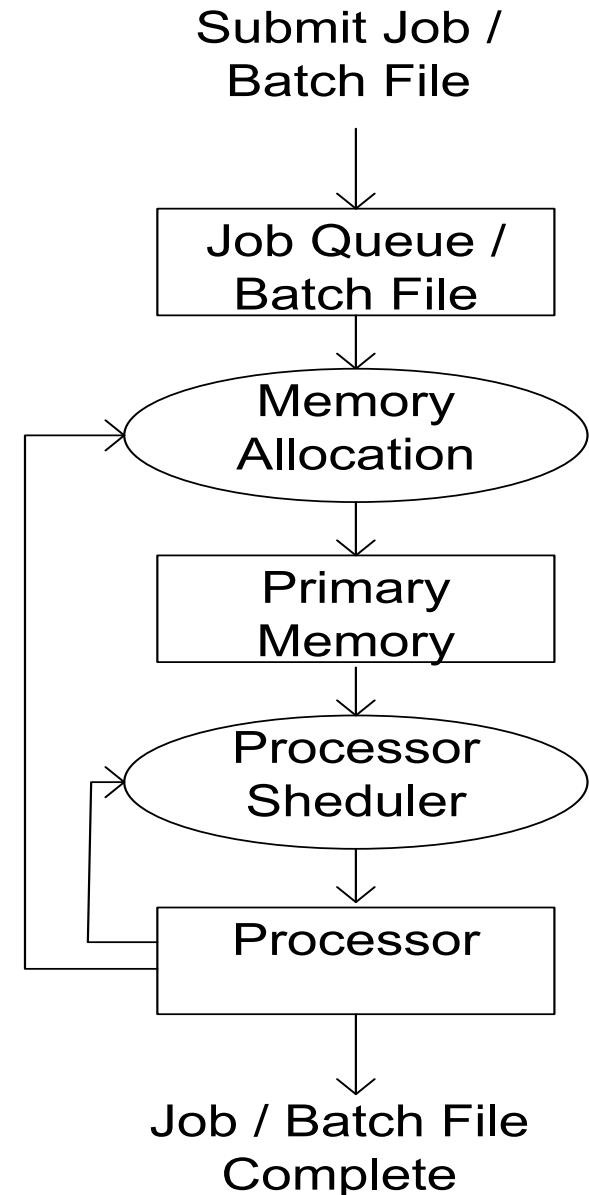
Operating Systems Evolution

- **Computers with no operating system**
 - Programming in machine language
 - Lack of I/O devices
- **Rudimentary OS**
 - Programming done in assembly
 - Some basic I/O devices
 - Some I/O control modules, assembler, debugger, loader, linker
- **Batch processing systems** – service a collection of jobs, called a batch, from a queue
 - Job – predefined sequence of commands, programs and data combined into a single unit
 - Job Control Language and monitor batch (interpreter for JCL)
 - The user doesn't interact with programs while they operate



Batch Systems

- Processor scheduling : FIFO
- Memory management:
 - Memory is divided in two parts: system memory and program memory (for programs)
- I/O management – no special problems, since a job has exclusive access to the I/O devices
- File management – present



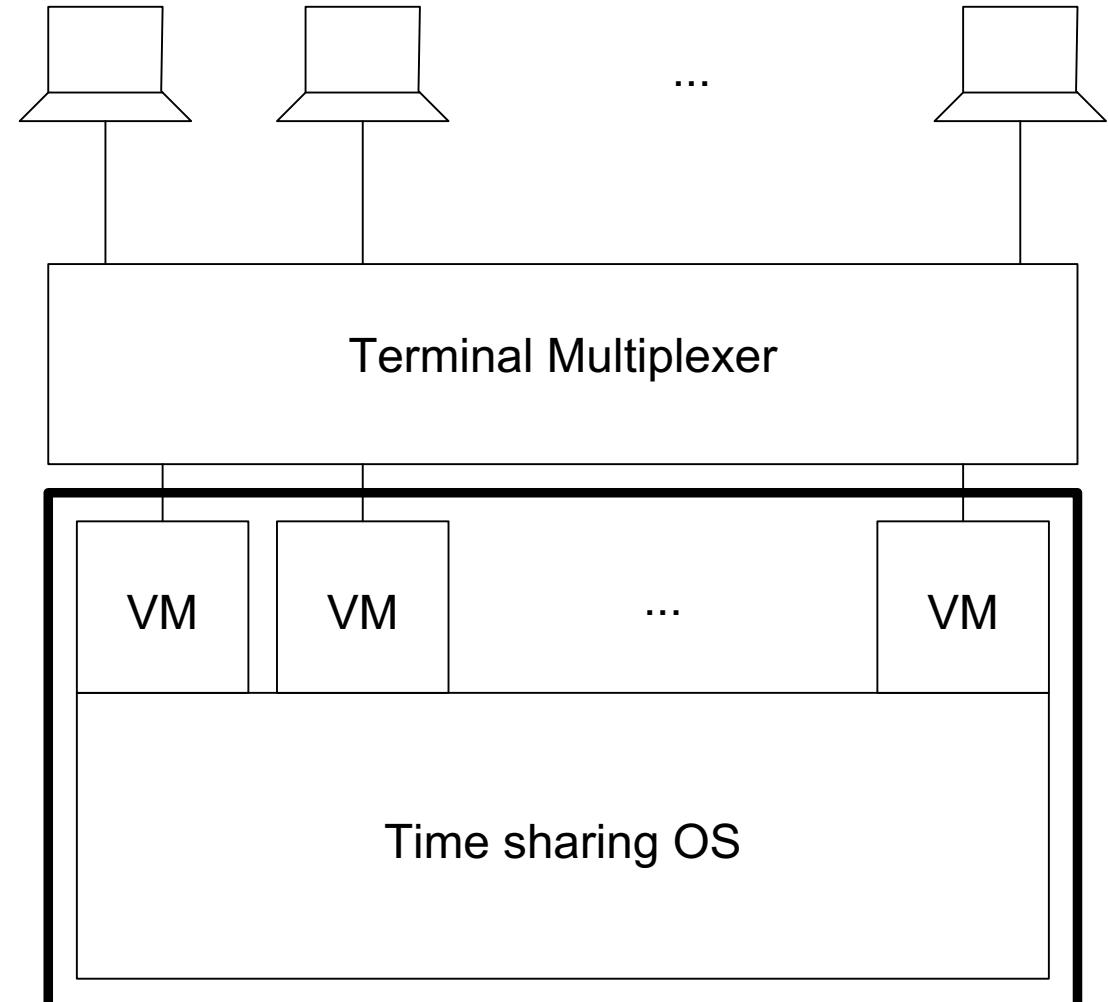
Operating Systems Evolution

- Operating systems using **multiprogramming**: the technique of loading multiple programs into space multiplexed memory while time-multiplexing the processor
 - Timesharing Systems
 - Real-time Operating Systems
 - Distributed Operating Systems
- Multiprogramming systems **common features**
 - Multitasking: multiple processes sharing machine resources
 - Hardware support for memory protection and I/O devices
 - Multi-user and multi-access support (through time sharing mechanisms)
 - Optional support for real time operations (based on efficient usage of multitasking support)
 - Interactive user interface



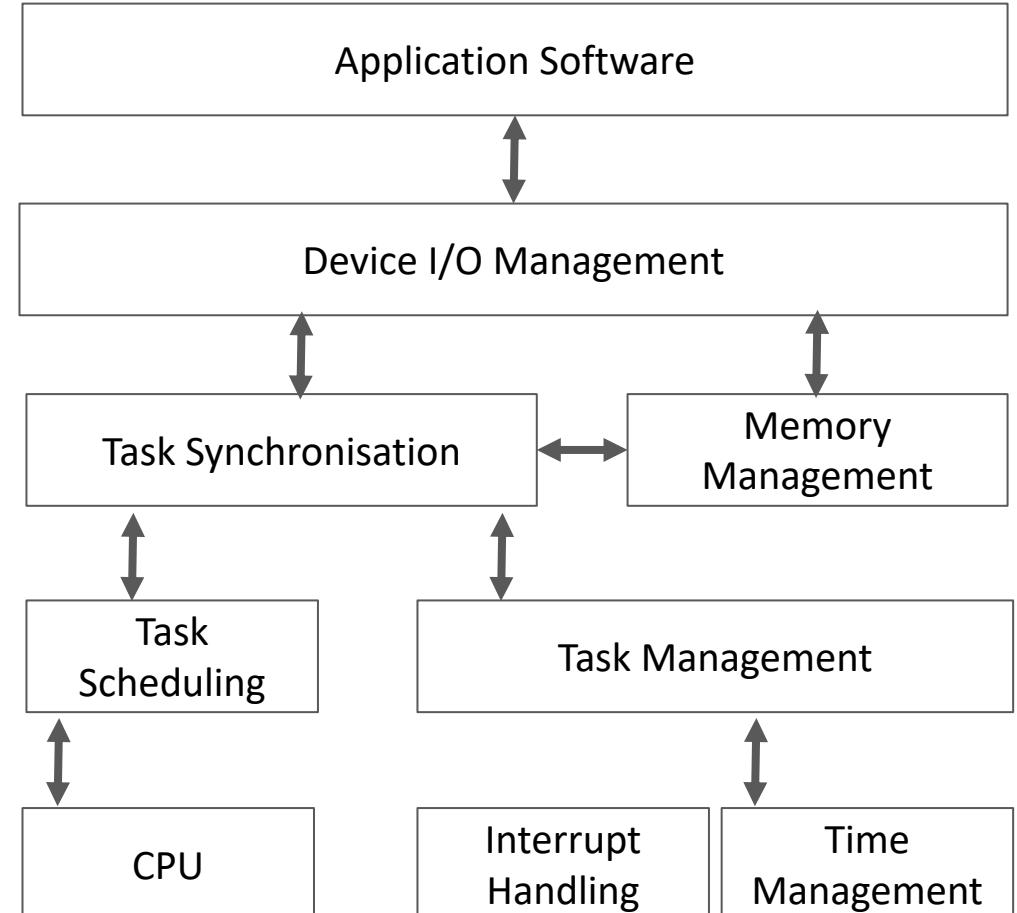
Time Sharing Systems

- Support for **multiprogramming** and **multi-user**
- Processor scheduling
 - Time slice (round robin)
- Memory management:
 - Protection and inter-process communication support
- I/O management
 - Support for protection and sharing between users
- File management
 - Protection support and sharing support between users



Real Time Operating Systems

- Used whenever a large number of **critical external events** have to be treated in a short or **limited interval of time**
- Support for multiprogramming/multi-tasking
- Main goal
 - **Minimisation of the response time** to service the external events



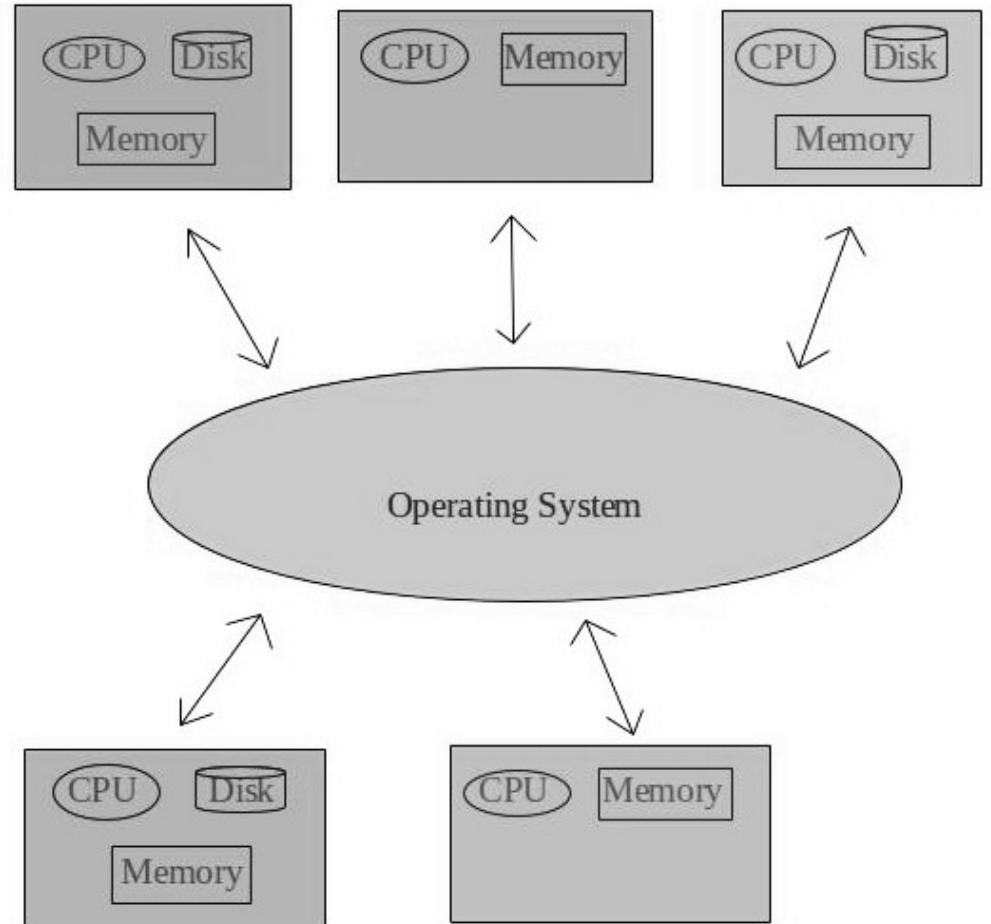
Real Time Operating Systems

- Processor scheduling:
- Priority based preemptive
- Memory management:
- Concurrent processes are loaded into the memory
 - Support for protection and inter-process communication
- I/O management:
- Critical in time
 - Processes dealing with I/O are directly connected to the interrupt vectors (for handling the interrupt requests)
- File management:
- It may be missing
 - If it exists, it should comply with requirements for timesharing systems and it should satisfy the requirements for real time systems

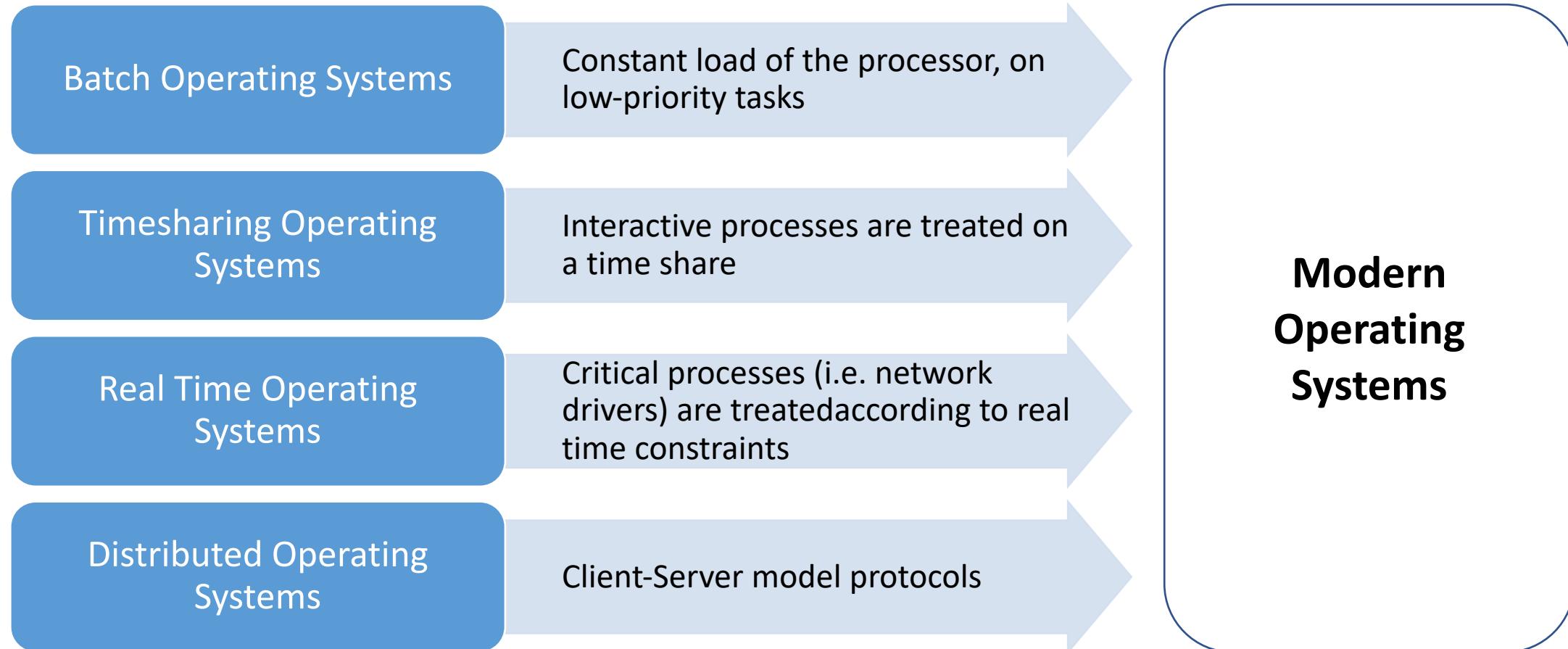


Distributed Operating Systems

- Multiprogramming induces a **strong centralisation** tendency
- Distributed OS aims for **decentralisation**
- Based on **computer network technologies**, with different communication and synchronization protocols
- **Client-server** application architecture
- **Security** and **protection** are the primary concerns



Modern Operating Systems



OS Implementation Considerations



OS Implementations

- **Monolithic Operating System**

- Try to achieve the functional requirements by executing all the code in the same address space to increase the performance of the system
- Too complex to manage

- **Hierarchical Operating System**

- Run most of their services in user space, aiming to improve maintainability and modularity of the codebase
- Suitable for Object Oriented Programming, the levels are very well defined

Implementation Considerations

- **Multi-programming:** the illusion that multiple programs are running simultaneously
- **Protection:** access to shared system resources
- **Processor modes:** different privilege levels
 - restrictions on operations that can be run
- **Kernels:** complete control over everything in the system (i.e., supervisor)



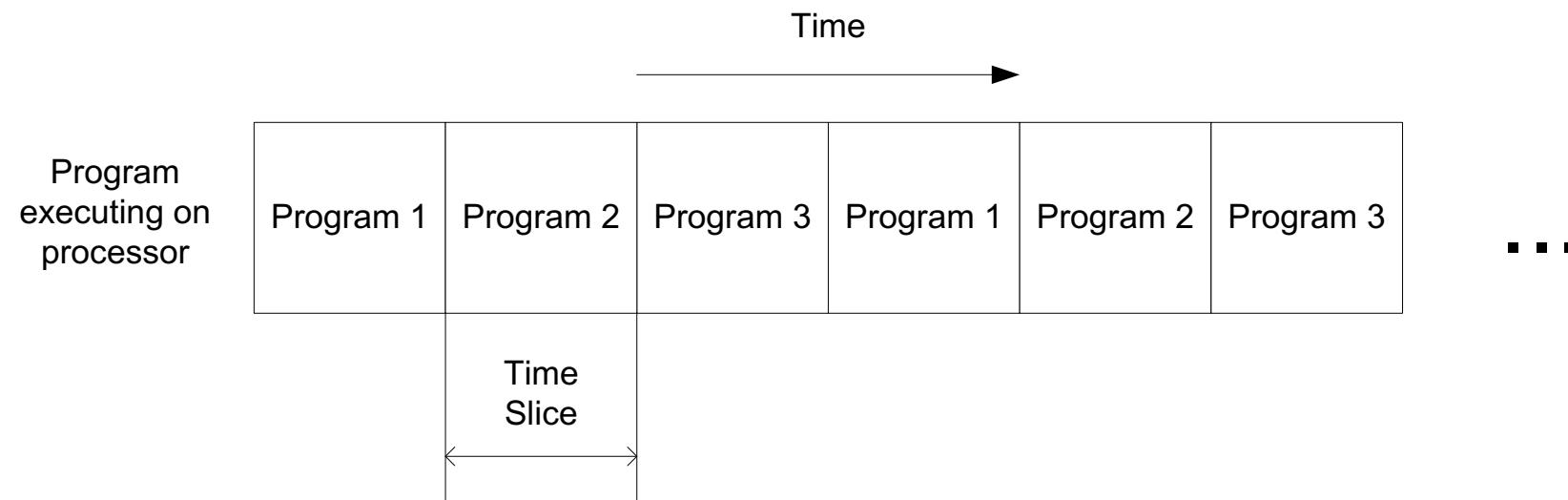
Multiprogramming (1)

- Technique that allows the system to present the illusion that multiple programs are running on the computer simultaneously
- **Protection** between programs is very important
 - Many multiprogrammed computers are **multiuser**
 - Allow multiple persons to be logged on at a time
- Beside protection, data **privacy** is also important
- Multiprogramming is achieved by **switching rapidly between programs**.
 - Each program is allowed to execute for a fixed amount of time – **timeslice**



Multiprogramming (2)

- When a program timeslice ends, the OS stops it, removes it and gives another program control over the processor – this is a ***context switch***
 - To do a context switch the OS copies the content of current program register file into memory, restores the contents of the next program's register file into the processor and starts executing the next program.
 - From the program point of view, they can't tell that a context switch has been performed

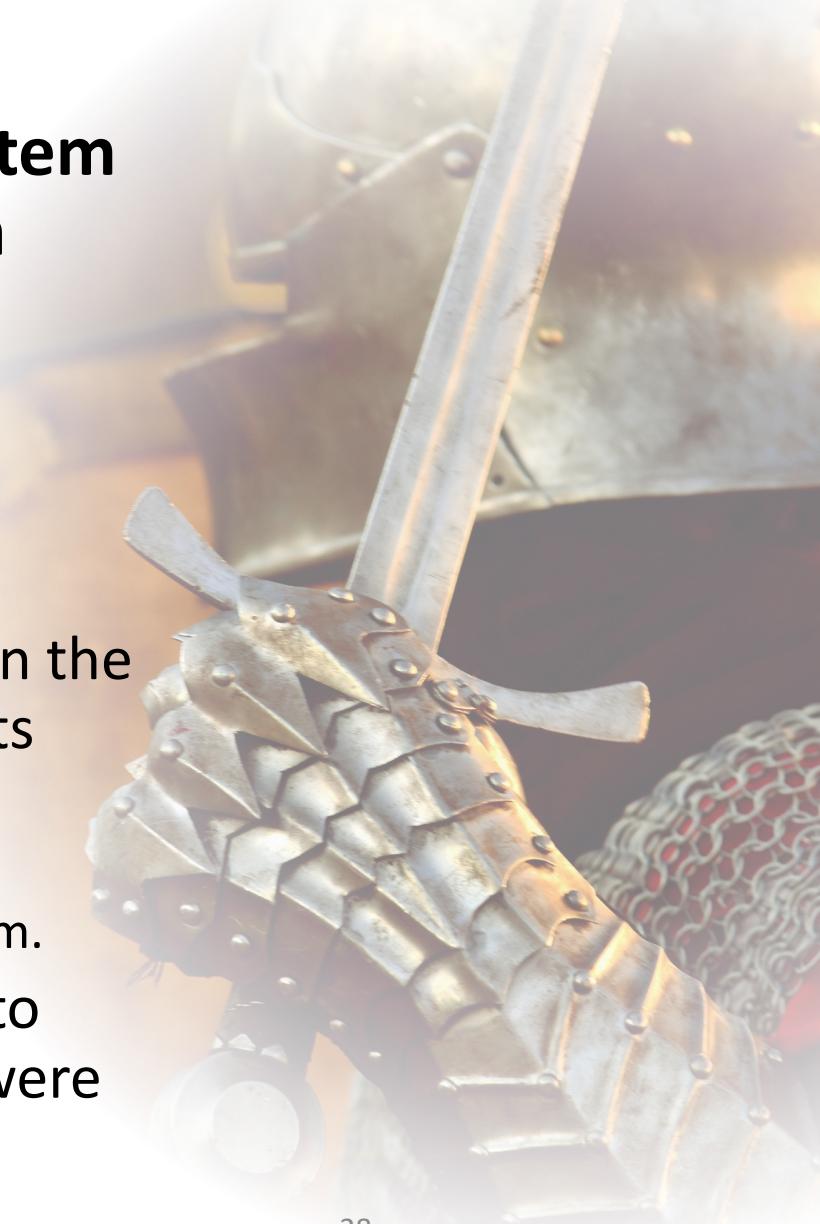


Protection (1)

- The result of any program running on a multiprogrammed computer must be the same as if the program was the only program running on the computer
- Programs must not be able to access other program's data and must be confident that their data will not be modified by other programs.
- Programs must not interfere with other program's use of I/O devices

Protection (2)

- Protection is achieved by having the **operating system have full control over the resources of the system** (processor, memory and I/O devices)
- ***Virtual memory*** is one of the techniques used to achieve protection between programs
 - Each program operates as if it were the only program on the computer, occupying a full set of the address space in its virtual space.
 - The OS is ***translating*** memory addresses that the program references into physical addresses used by the memory system.
 - As long as two program's addresses are not translated to same address space, programs can be written as they were the only ones running on the machine



Privileged Mode

- To ensure that the OS is the only one that can control the physical resources it executes in **privileged mode**
- OS is also responsible for **low level UI**
 - Keys are pressed, the OS is responsible to determine which program should receive the input
 - When a program wants to display some output, the user program executes some system call that displays the data
- User programs execute in **user mode**
 - When user mode programs want to execute something that requires privileged rights, it sends a request to the OS, known as **system call**, that asks the OS to do the operation for them

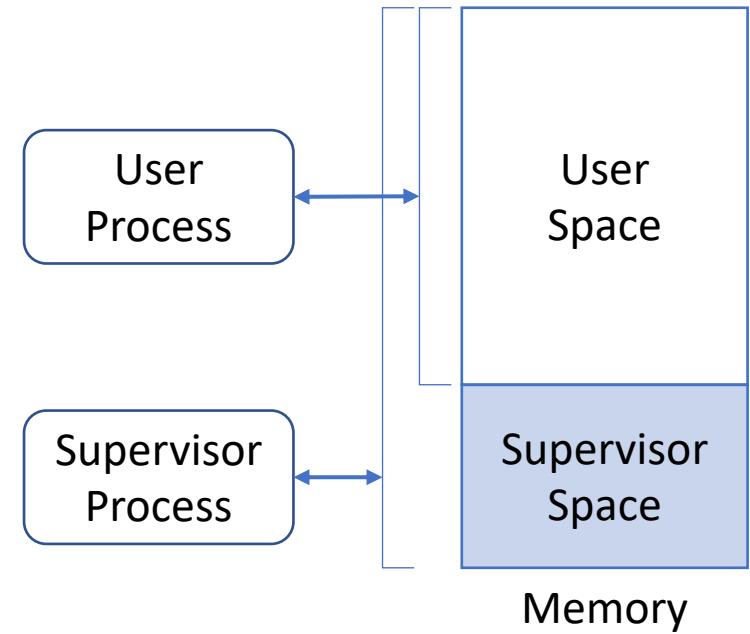


Processor Modes (1)

- Processor Modes are operating modes for the CPU that **place restrictions** on the operations that can be performed by the currently running process
- Hardware supported CPU modes help the operating system to **enforce rules** that would prevent viruses, spyware, and/or similar malware to run
 - Only very specific and limited “kernel” code would run unrestricted.
 - Any other software (including portions of the operating system) would run restricted and would have to ask the “kernel” for permission to modify anything that could compromise the system.
- Multiple mode levels could be designed.

Processor Modes (2)

- Mode bit to define **execution capability** of program on a processor
- *Supervisor mode*
 - The processor can execute any instruction
 - Instructions that can be executed only in supervisor mode are called *supervisor, privileged or protected* instructions (e.g., I/O instructions)
 - Execution process has access on both memory spaces
- *User mode*
 - The processor can execute a subset of the instruction set
 - Executing process has access only to the user space
- Some microprocessors do not make a difference between protected and user mode
- The mode bit may be logically extended to define areas of memory to be used when the processor is in supervisor mode versus when it is in user mode



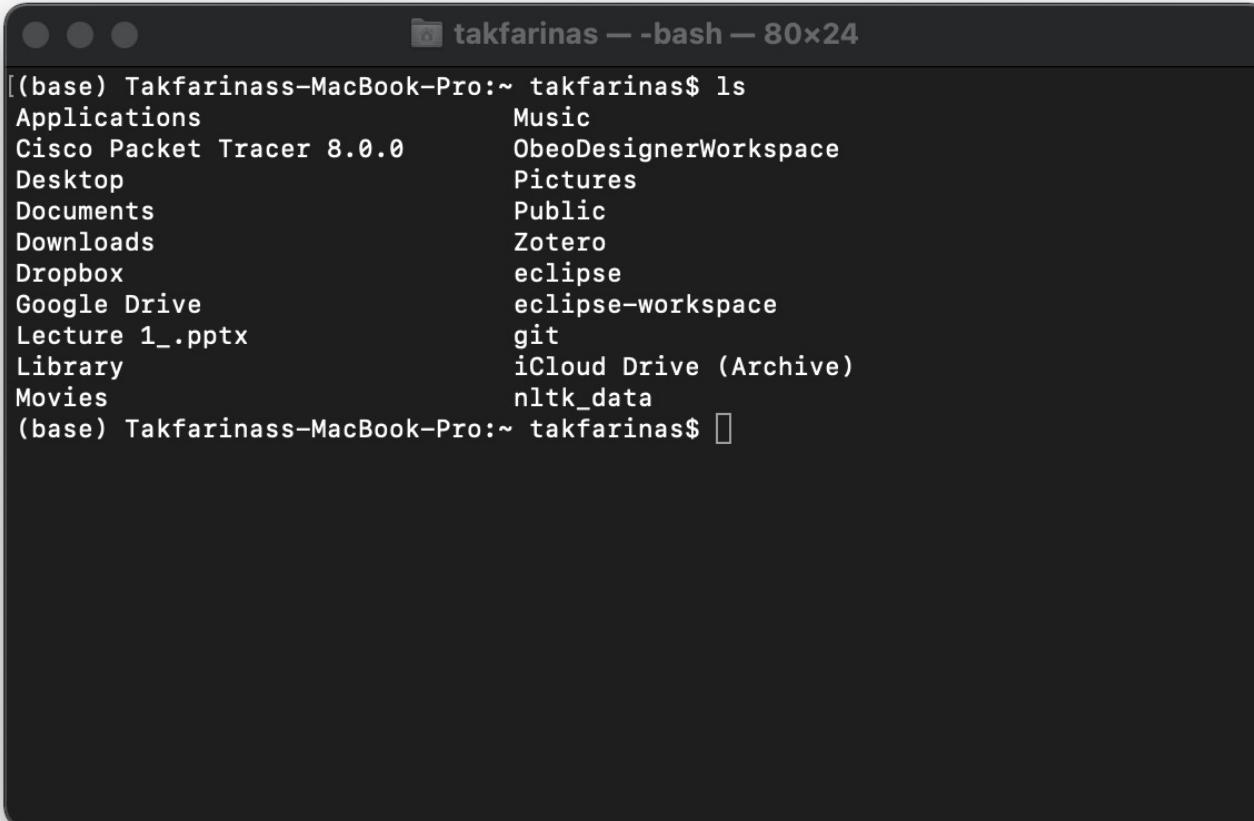
Kernel

- The part of the operating system that executes in supervisor mode is called **kernel** or **nucleus**
- Operates as **trusted** software
 - Implements protection mechanisms that could not be changed through the actions of un-trusted software executing in user mode
 - Provides the lowest level abstraction layer for resources (memory, processors and IO devices)
- Fundamental design decision: should a given function of the OS be incorporated in the kernel or not?
 - Protection issues
 - Performance issues



Methods for Requesting System Services

- Through command line interface
 - By calling a specific command
 - Using a command interpreter known as a shell
- From user processes requesting services from OS:
 - By calling a system function
 - By sending a message to a system process



```
takfarinas — bash — 80x24
(base) Takfarinass-MacBook-Pro:~ takfarinas$ ls
Applications           Music
Cisco Packet Tracer 8.0.0  ObeoDesignerWorkspace
Desktop                Pictures
Documents               Public
Downloads              Zotero
Dropbox                 eclipse
Google Drive            eclipse-workspace
Lecture 1_.pptx         git
Library                 iCloud Drive (Archive)
Movies                  nltk_data
(base) Takfarinass-MacBook-Pro:~ takfarinas$ 
```

Command execution mechanism

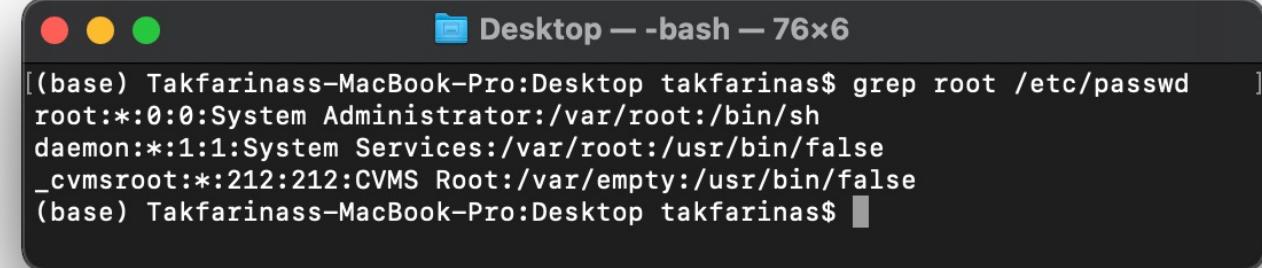
- A key pressed by the user generates a hardware interrupt
- A specialised module of the OS reads the keyed character and then stores it in a special command line buffer
 - There are special characters (i.e., to edit the command line, that are not stored in the command line buffer)
- End of line detected: control taken by the command interpreter (shell):
 - Analysis of the command (with error or success)
 - If success, then the command interpreter decides if it is about an internal or external command (for another module)
 - If internal command: tries the execution that can end successfully or with error
 - If external: looks for the corresponding executable file and executes it with the detected parameters from previous phase



Command execution example

- Semantics of **grep** establish that the first string parameter (first) represents the search target, while the second parameter represents a file name (where to search)

```
>$ grep mouse mouse.txt
```

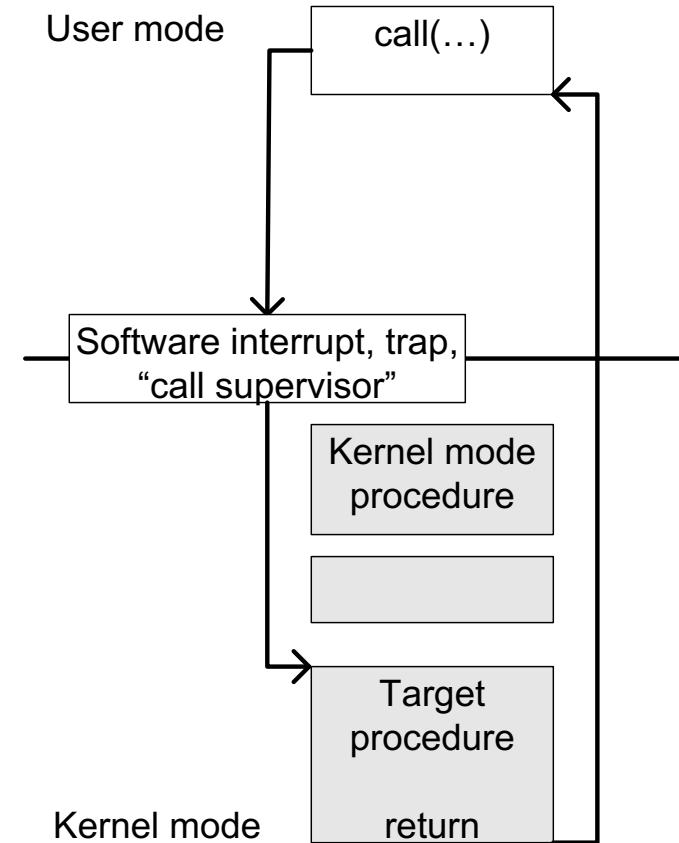


The screenshot shows a terminal window on a Mac OS X desktop. The title bar reads "Desktop — bash — 76x6". The terminal content displays the results of a grep search for the string "mouse" in the file "mouse.txt". The output is as follows:

```
(base) Takfarinass-MacBook-Pro:Desktop takfarinas$ grep mouse mouse.txt
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_cvmsroot:*:212:212:CVMS Root:/var/empty:/usr/bin/false
(base) Takfarinass-MacBook-Pro:Desktop takfarinas$
```

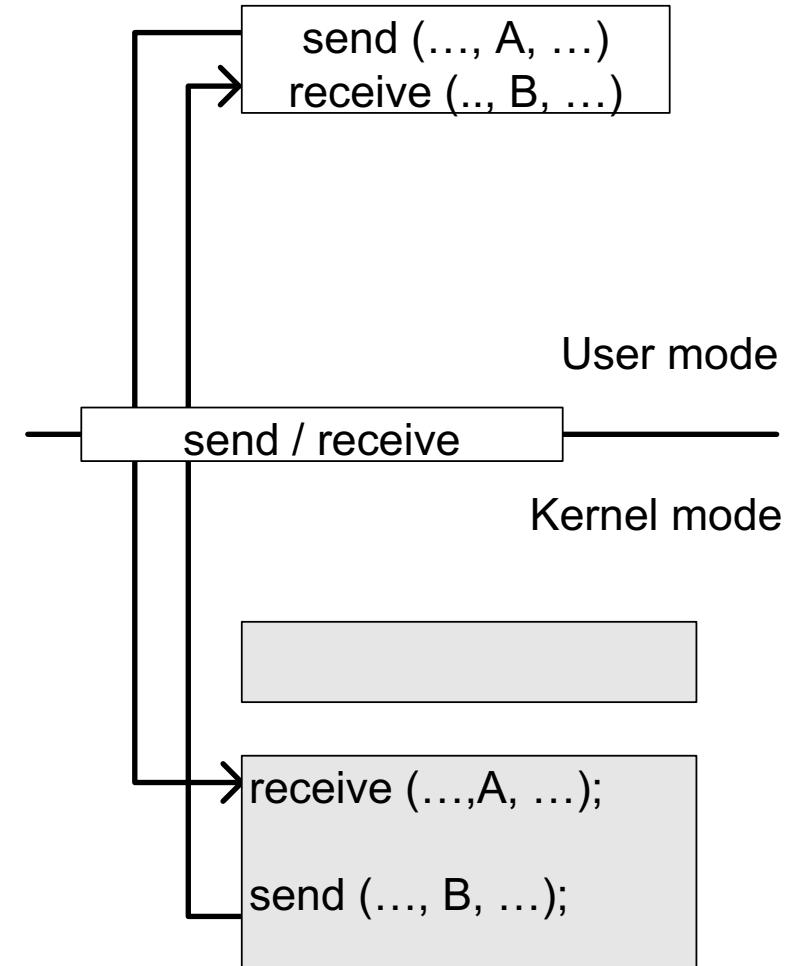
System Call

- The parameters of the call are passed according to the **specific OS convention** and hardware architecture
- Switch to **protected (supervisor) mode** using a specific mechanism
 - E.g., software interrupt, trap, special instruction of type “call supervisor”
 - mechanism that is different from a normal call
- A **special module** takes over, that will analyse the parameters and the access rights
 - This module can reject the system call
- If accepted: the **corresponding routine** from the OS is executed and the **result** is returned to the user
 - upon return, the user mode is restored



Messages

- User process **constructs a message** that describes a desired service (A)
- Uses the **send** function to pass the message to a trusted operating system process
 - The send function checks the message
 - switches the processor to **protected mode**
 - and then delivers the message to the process that implements the target function
- Meanwhile, the user waits for result with a **message receive** operation.
- When the **kernel finishes processing the request**, it sends a message (B) back to the user process



References

- “Operating Systems – A modern perspective”, Garry Nutt, ISBN 0-8053-1295-1
- Funny video: <https://youtu.be/aJFwVOW0Nww>

